

Isabelle: the last 40 years (and the next)

Lawrence C Paulson

Isabelle Workshop, 24 July 2026

Yesterday

What was the big idea back in '86?

- ❖ *At the start: LCF, logic rules coded as functions in ML*
- ❖ *Aiming to build something like LCF, but faster and better — for Martin-Löf type theory*
- ❖ *Prolog-like backtracking with variables, higher-order unification, also search via lazy lists*
- ❖ *But a syntax based on typed λ -calculus can support many logics: a logical framework*

General milestones

- ❖ **1986** first distribution (MLTT, LJ, LK); Tobias joins the project
- ❖ **1989** intuitionistic HOL as the *logical framework* for natural deduction
- ❖ **1991** Isabelle / ZF
- ❖ **1993** Isabelle / HOL
- ❖ **1999** locales; Makarius joins
- ❖ **1999** Proof General
- ❖ **2002** code generation
- ❖ **2008** multicore
- ❖ **2011** Isabelle / jEdit PIDE
- ❖ **2018** Isabelle / VS Code*

1989: the type of natural numbers

("N_form", "[| N type |]"),

("N_intr0", "[| 0 : N |]"),

("N_intr_succ", "[| a : N |] ==> [| succ(a) : N |]"),

("N_elim",

 "[| p : N |] ==> [| a : C(0) |] ==> \

\ (! (u) [| u : N |] ==> ! (v) [| v : C(u) |] \

\ ==> [| b(u,v) : C(succ(u)) |]) ==> \

\ [| rec(p,a,b) : C(p) |]"),

Specification milestones

- ❖ **1991** order-sorted polymorphism
- ❖ **1994** (co)inductive definitions
- ❖ **1997** axiomatic type classes
- ❖ **1999** Isar proof language
- ❖ **2008** nominal logic (to formalise variable binding)
- ❖ **2009** (partial) recursive function definitions
- ❖ **2012** compositional (co)datatypes based on BNFs

Heuristic automation milestones

- ❖ **1989** simplifier, prover on Isabelle's native "Prolog"
- ❖ **1997** a tableau prover coded in ML: **blast**, **auto**
- ❖ **2008** Sledgehammer, the best "hammer" out there
- ❖ **2010** Nitpick counterexample finder
- ❖ **2011** Adding SMT solvers to Sledgehammer; later, higher-order tools (Zipperposition)

Decision procedure milestones

- ❖ **1999** basic linear arithmetic
- ❖ **2005** Cooper's algorithm by *reflection*
- ❖ **2011** Gröbner basis algorithms for algebra
- ❖ **2016** argo reflected SMT solver
- ❖ **2019** verified real asymptotics

Formalised mathematics milestones

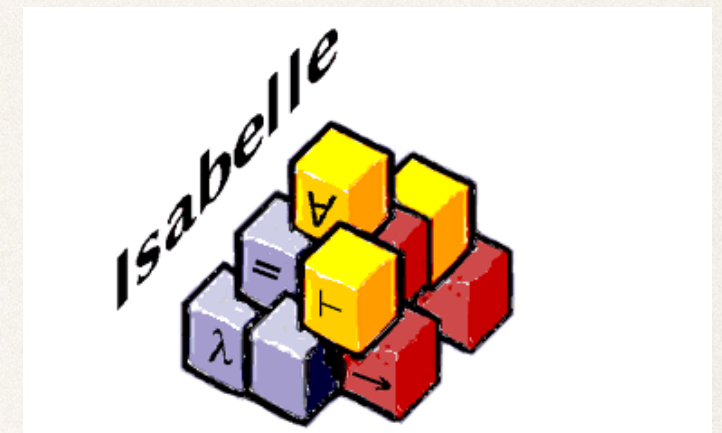
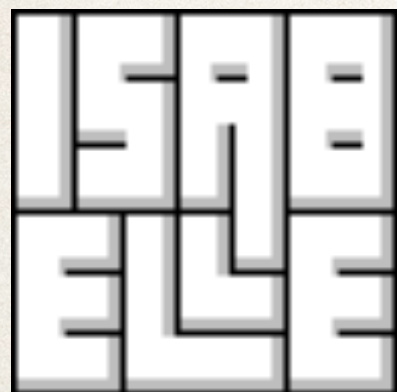
- ❖ 1996 ZF set theory as “computational logic”; later, results on AC, Gödel’s constructible universe, forcing
- ❖ 1998 real analysis, incl non-standard analysis, parts of Newton’s *Principia*; later, a vast analysis library
- ❖ 1999 abstract algebra, now to Galois theory, etc.
- ❖ 2006 “Tame graphs” for Kepler Conjecture verification
- ❖ 2017–26 deep, recent results mostly in combinatorics

```

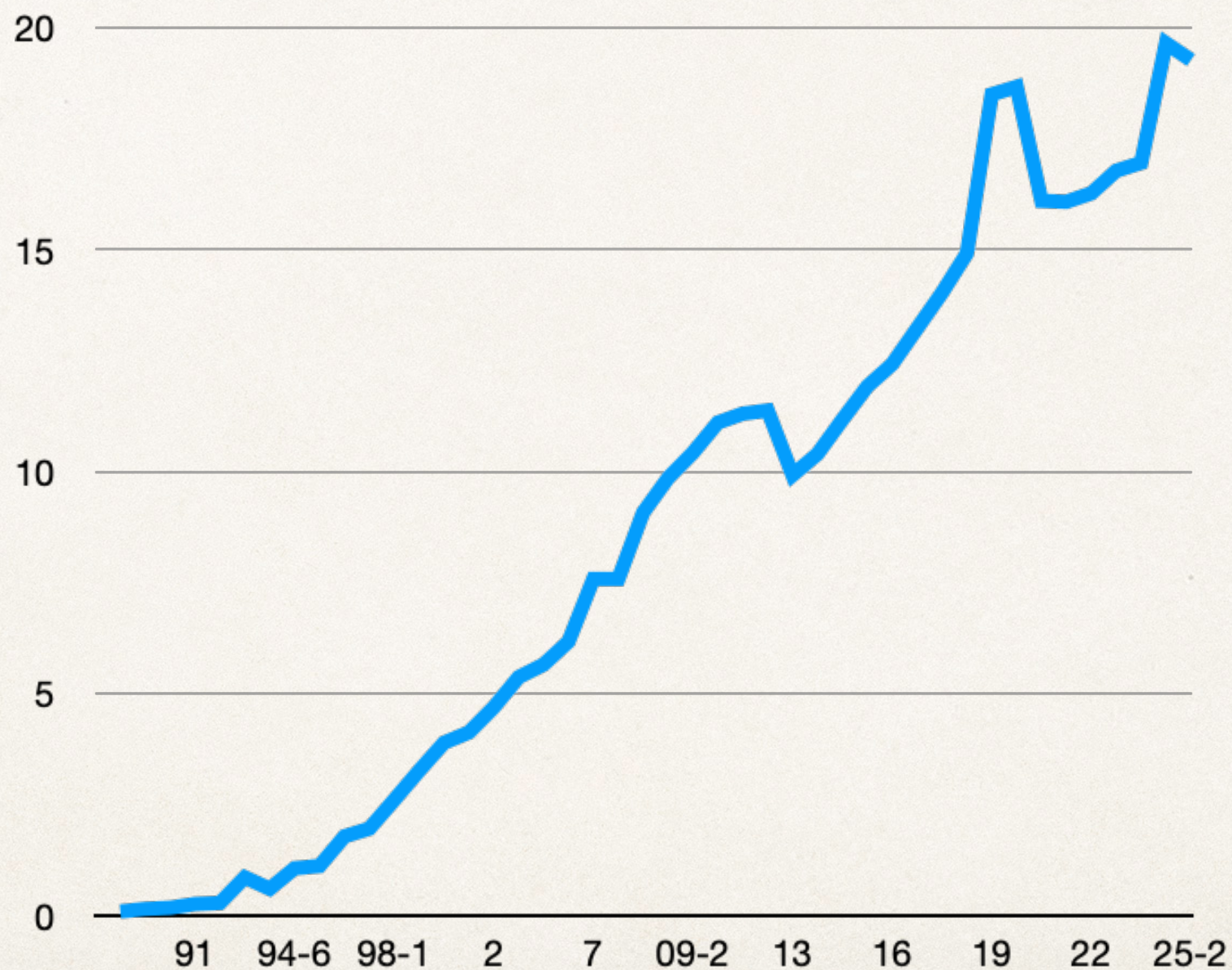
goalw thy [AC16_def,W04_def]
      "!!n k. [| AC16(k #+ m, k); 0 < k; 0 < m; k:nat; m:nat |] ==> W04(m)";
by (rtac allI 1);
by (excluded_middle_tac "Finite(A)" 1);
by (rtac lemma1 2 THEN REPEAT (assume_tac 2));
by (resolve_tac [lemma2 RS revcut_rl] 1);
by (REPEAT (eresolve_tac [exE, conjE] 1));
by (eres_inst_tac [("x","A Un y")] allE 1);
by (forward_tac [infinite_Un] 1 THEN (mp_tac 1));
by (REPEAT (eresolve_tac [exE, conjE] 1));
by (resolve_tac [well_ord_LL RS exE] 1 THEN REPEAT (assume_tac 1));
by (fast_tac (AC_cs addSIs [nat_succI, add_type]) 1);
by (res_inst_tac [("x","ordertype(LL(T, succ(k), y), x)")] exI 1);
by (res_inst_tac [("x","lam b:ordertype(LL(T, succ(k), y), x). \
\      (GG(T, succ(k), y)) ` \
\      (converse(ordermap(LL(T, succ(k), y), x)) ` b)")] exI 1);
by (simp_tac AC_ss 1);
by (fast_tac (empty_cs addSIs [conjI, lam_funtype RS domain_of_fun]
      addSEs [Ord_ordertype, all_in_lepoll_m, OUN_eq_x]) 1);
qed "AC16_W04";

```

The Isabelle logo contest (1997)



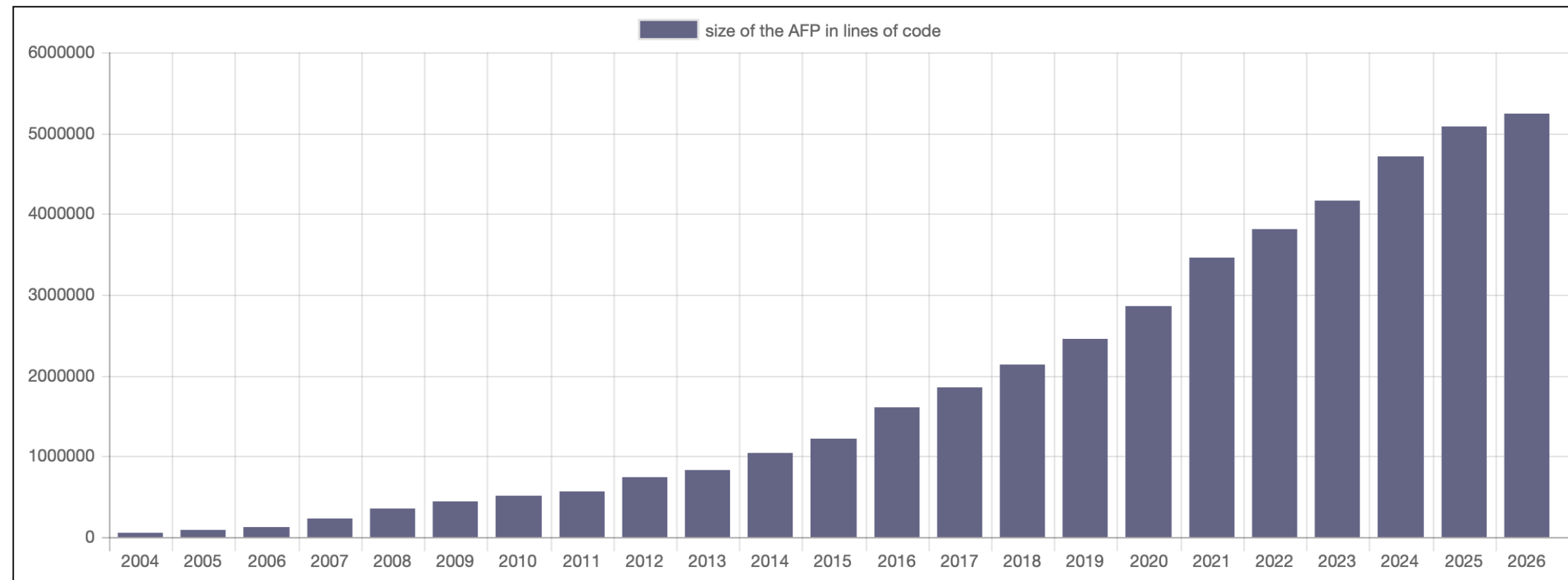
Size of the distribution in MB



Archive of Formal Proofs

Founded in 2004; now over 1000 contributions!

Growth in lines of code:



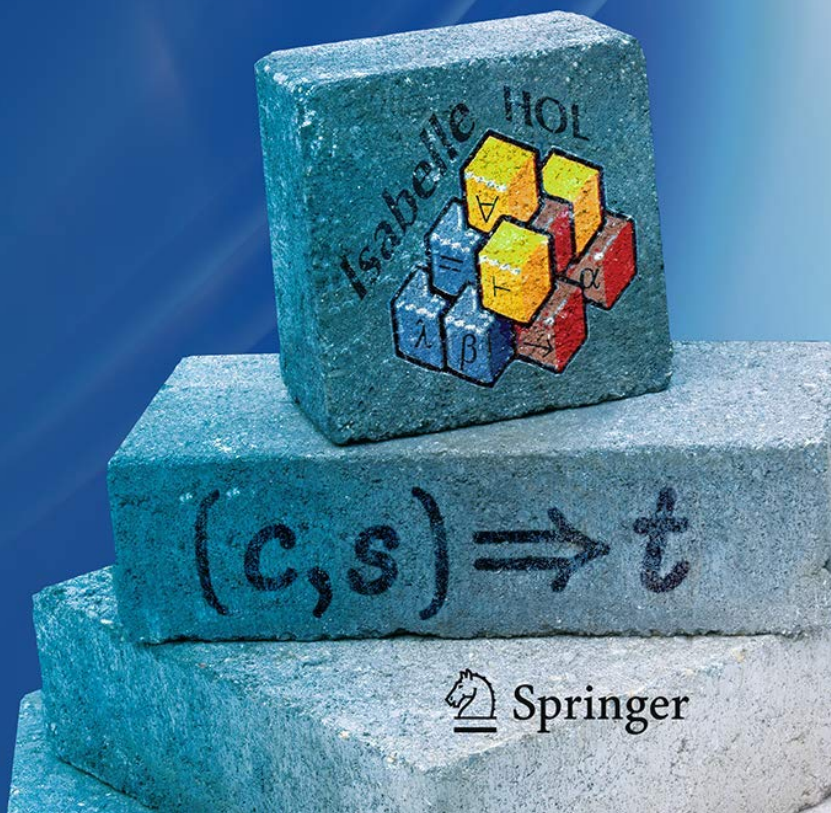
Verification milestones

- ❖ Semantics of Java (well, Jinja) (also with threads)
- ❖ Formalised PL theory, e.g. *Concrete Semantics*
- ❖ *Functional Data Structures and Algorithms*
- ❖ The seL4 microkernel & the AutoCorres C verifier
- ❖ Amazon's *Nitro Isolation Engine*; Apple *corecrypto*

Tobias Nipkow · Gerwin Klein

Concrete Semantics

With Isabelle/HOL



Springer

Functional Data Structures and Algorithms

A Proof Assistant Approach

Tobias Nipkow (Editor)

AVL
red-black
2-3
splay

induction
correctness
complexity

α - β pruning

string search

binomial

heap

pairing

skew



ASSOCIATION FOR COMPUTING MACHINERY

Today: formalised mathematics

Maths a few years ago (2017)

- ❖ Lots formalised already

Matrix theory, e.g. Perron–Frobenius

- ❖ But... was it *deep* enough?
Are there **no-go** areas?

Analytic number theory, e.g.
Hermite–Lindemann

Homology theory

- ❖ ALEXANDRIA project: to
explore mathematics in
practice (2017–23)

Measure, integration
and probability theory

Complex analysis: residue
theorem, prime number theorem

By 2019: algebraic closure of a field

Every field admits an algebraically closed extension

(Example: adjoining a root of $x^2 + 1$ to $\mathbb{R}$ to get $\mathbb{C}$)

In general, a *limit* of field extensions

$$K = E_0 \rightarrow E_1 \rightarrow E_2 \rightarrow \cdots \rightarrow E_n \rightarrow \cdots$$

obtained by adjoining roots. We can form this limit using Zorn's lemma

The work of two summer students, Paulo de Vilhena and Martin Baillon, and the first formalisation of this result in any system.

By 2022: additive combinatorics

The study of the *additive structure of sets*, combining ideas from many branches of mathematics

Plünnecke–Ruzsa inequality:
an upper bound on $|mB - nB|$

Khovanskii's theorem: $|nA|$ grows like a polynomial for sufficiently large n

Kneser's theorem and the Cauchy–Davenport theorem: lower bounds for $|A + B|$

Balog–Szemerédi–Gowers: a deep result bearing on Szemerédi's theorem

Lessons from ALEXANDRIA

- ❖ You can formalise just about anything using **locales** and **simple types**
- ❖ **Set theory** can be added cleanly when required
- ❖ Our proofs are **legible** and **performant**
- ❖ We formalised the work of two Fields medalists (Roth, Gowers), an Abel prize winner (Szemerédi) ... and the legendary Paul Erdős too.

Today: industrial scale proofs

seL4 Home ↩

What is seL4?

A high-assurance, high-performance operating system microkernel

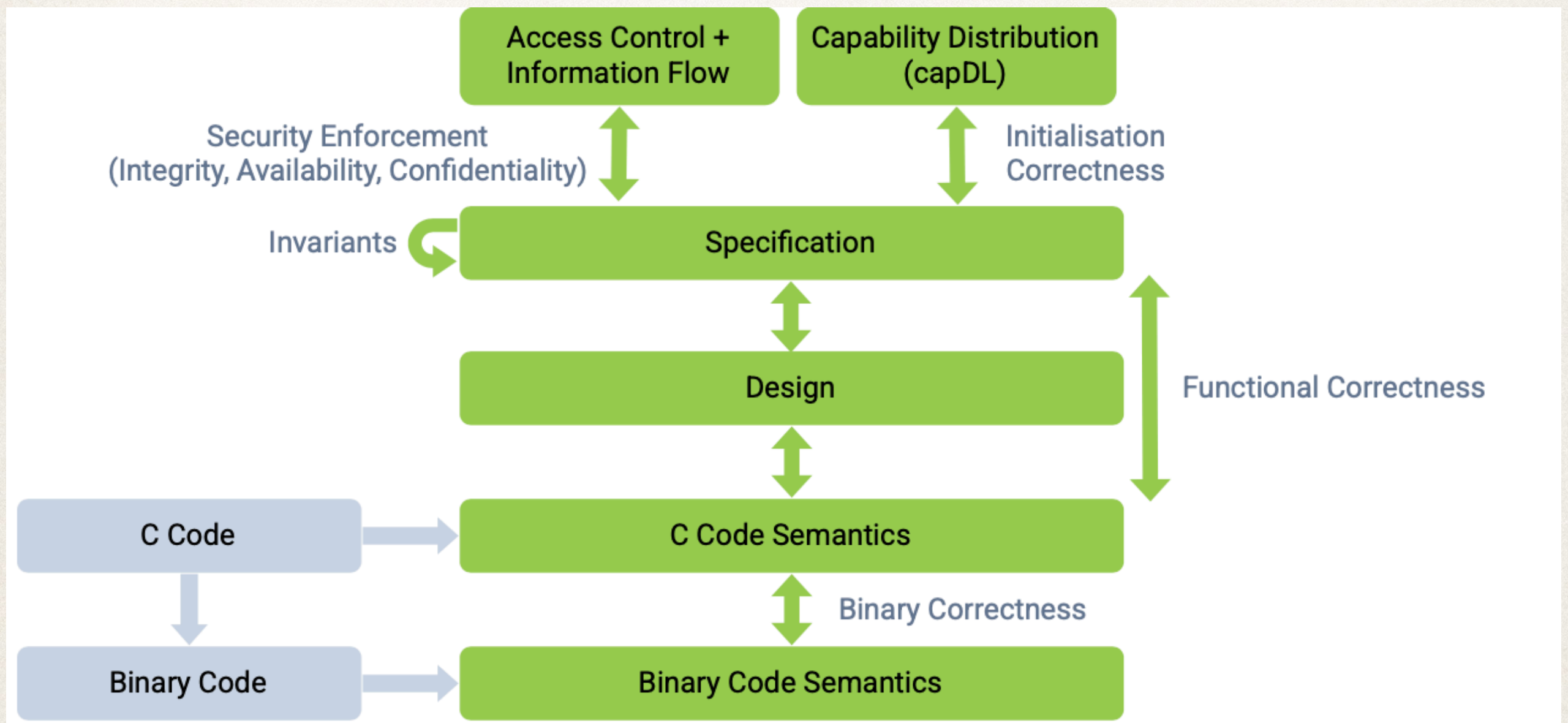
seL4 is unique because of its comprehensive formal verification.

seL4 is open source, available on GitHub, and supported by the seL4 Foundation.

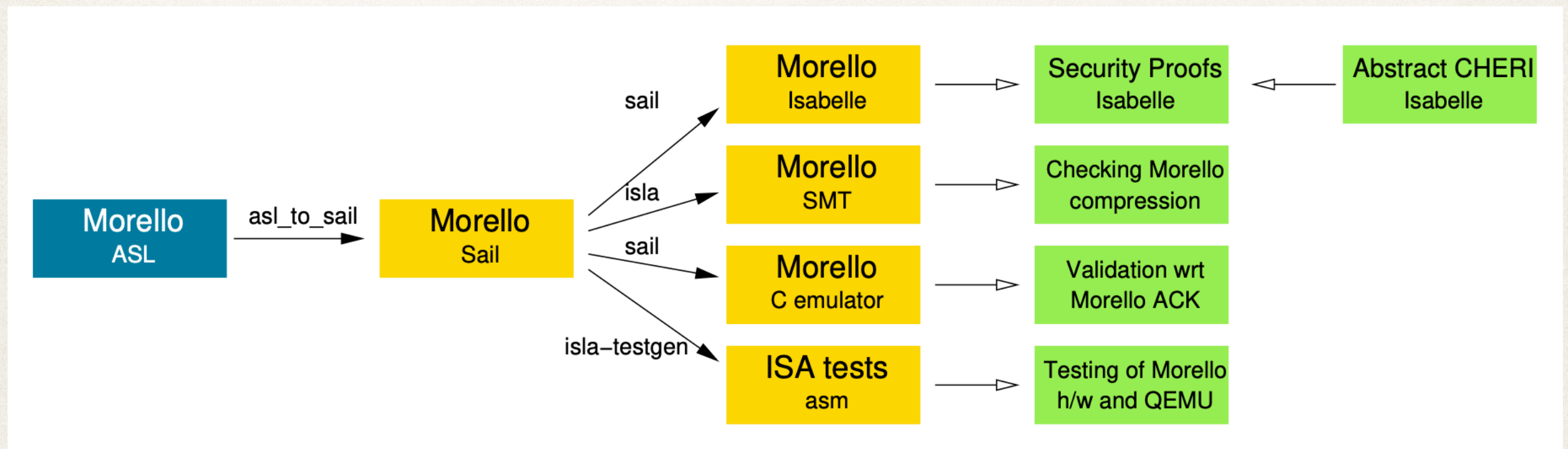
The foundation for trustworthy systems.

“in the entire 10,000 lines of verified code nothing has gone wrong in all that time”

seL4 proofs: right down to binary



Morello CHERI-based architecture

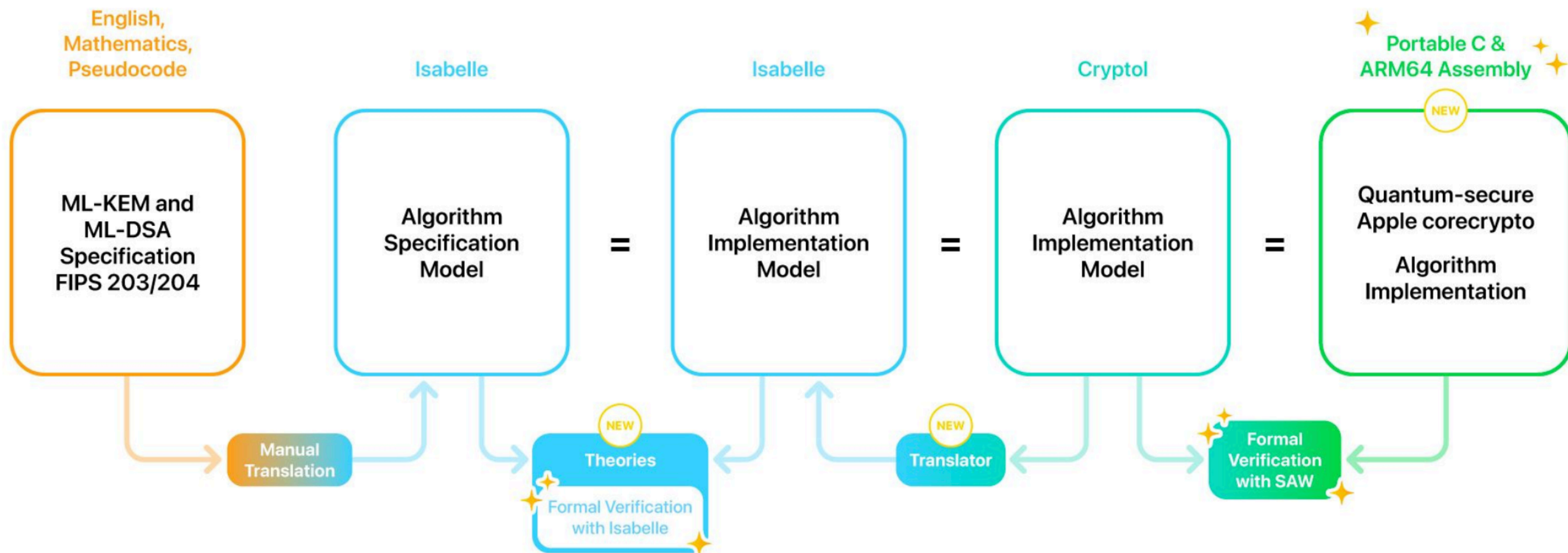


CHERI: hardware-supported *capabilities* to enable fine-grained memory protection. *Morello*: Arm's CHERI-enabled prototype architecture.

“The proof consists of around 37000 generated lines, 8600 manually written lines, as well as 8900 lines for the abstract model, monotonicity proof, and proof tools. The proof executes in 7hrs 20mins CPU time on an i7-10510U CPU at 1.80GHz, but only 3hrs 23mins real time thanks to parallel execution, with peak memory consumption of 18GB.”

*Plus an earlier, large Isabelle verification by
Kyndylan Nienhuis*

A New Approach to Formal Verification of Apple corecrypto



We designed a custom formal verification approach for Apple corecrypto to prove that our new quantum-secure implementations of ML-KEM and ML-DSA algorithms are equivalent to their specification. We first verify our portable C implementations step-by-step, then we verify our ARM64 assembly implementations by showing them to be functionally equivalent to the portable C.

“First, we manually translate our portable C implementation of each algorithm into the Cryptol language.

To prove that our Cryptol model is equivalent to the specification, we ... generate a model of the implementation in Isabelle, directly from the Cryptol.

The work involved is considerable, requiring more than 50,000 proof steps.”

Amazon Web Services

Automated reasoning

EC2's formally verified “isolation engine” provides mathematical assurance of virtual-machine isolation

Splitting the “separation kernel” off from the rest of the Nitro security system and using only a subset of the Rust programming language to code it enabled its formal verification.

By [Dominic Mulligan](#), [Nathan Chong](#)

June 10, 2026 | 7 min read

 [Share](#)

Notes on the Nitro Isolation Engine

- ❖ Coded in Rust; Isabelle's *AutoCorrode* package supports μ Rust
- ❖ Separation logic for specs; WP-calculus for proofs
- ❖ “330,000 lines of machine-checked mathematics”, proving...
 - ❖ “**Confidentiality and integrity:** Only authorized information flows can occur.”
 - ❖ “**Functional correctness:** The implementation behaves exactly as specified.”
 - ❖ **Absence of runtime errors; memory safety.**

Now, what about AI?

Things Sledgehammer can't do

- ❖ Communicate in natural language
- ❖ Deliver a proof skeleton *containing gaps*
- ❖ Tell you that *your goal is false* and supply a fix
- ❖ Write **legible** proofs, using reasonable names
- ❖ Working through a task writing 1000s of lines

Formal proof: the *best* use for LLMs

- ❖ Interactive proof requires scarce, expensive talent
- ❖ LLMs are *extremely good* at writing proofs
 - ❖ a feedback loop to catch errors
 - ❖ excellent general knowledge of mathematics
 - ❖ translating mathlib and HOL Light to idiomatic Isabelle
 - ❖ even including comments!

text <The mean value theorem turns the derivative bound into a Lipschitz bound.>

lemma cos_contraction_lt:

fixes $x\ y :: \text{real}$

assumes " $x < y$ " " $|x| \leq 1$ " " $|y| \leq 1$ "

shows " $|\cos x - \cos y| \leq \sin 1 * |x - y|$ "

proof -

have cont: "continuous_on $\{x..y\}$ cos"

by (intro continuous_intros)

have deriv: " $((\cos :: \text{real} \Rightarrow \text{real}) \text{ has_derivative } (*) (- \sin u)) (\text{at } u)$ " **for** $u :: \text{real}$

using DERIV_cos[of u] **unfolding** has_field_derivative_def **by** simp

then obtain ξ **where** ξ : " $\xi \in \{x <..<y\}$ " " $|\cos y - \cos x| \leq |\sin \xi| * |y - x|$ "

using mvt_general[OF $\langle x < y \rangle$ cont deriv] **by** (auto simp: abs_mult)

have " $|\xi| \leq 1$ "

using ξ assms **by** auto

have " $|\cos y - \cos x| \leq |\sin \xi| * |y - x|$ "

using ξ **by** (simp add: abs_mult)

also have " $\dots \leq \sin 1 * |y - x|$ "

by (simp add: $\langle |\xi| \leq 1 \rangle$ abs_sin_le_sin1 mult_mono')

finally show ?thesis

by (simp add: abs_minus_commute)

qed

Classic AI problems, solved!

- ❖ *Planning* of a large proof development
- ❖ *Common sense*: the LLM says " $y < x$ looks wrong; didn't you mean $x < y$?"
- ❖ *Translation*: **you** can say "Port the proof from mathlib, like you did last time."
- ❖ *Reasoning by analogy*: **you** can say "use the approach found in theory T , except don't do X ."

The Becker–Mulligan AI Toolset

- ❖ *I/Q*: experimental Isabelle / jEdit plugin providing an MCP server, letting MCP-capable agents do interactive proofs, autonomously or collaboratively.
- ❖ *I/R* (Isabelle / REPL): interactive theory exploration outside jEdit, from the command line or programmatically via MCP. Crude but effective.

Free download <https://github.com/aws-labs/AutoCorrode>

AI impacts involving Isabelle

- ❖ J Urban: autoformalisation of *Topology* by Munkres
- ❖ numerous experiments with AI tools
 - ❖ formalisation of four of the “100 theorems”
 - ❖ creation of a whole new Algebra library
- ❖ Zillions of AI-generated AFP submissions

And AI's drawbacks?

- ❖ cost and environmental concerns
- ❖ slow response times *vs* auto-mode running wild
- ❖ over-detailed proofs (with excessive comments)
- ❖ the risk of incorrect theorem statements or definitions
- ❖ experts need to review these proofs: a new bottleneck

AI agents are dumber than you think!

Isabelle is ideal for the AI age

- ❖ *Automation* for shorter and clearer proofs
- ❖ *Structured proofs*
 - ❖ Perfect for top-down proof planning
 - ❖ **Legibility**: vital if AI is writing the proofs
- ❖ *Sledgehammer* to repair or simplify AI slop

Looking to the future

Engaging with AI

- ❖ Seeking the best tooling (Interactive? Autonomous?)
- ❖ Adapting to a new, rapidly changing age
 - ❖ *How do we teach?*
 - ❖ *When is a formalisation novel research?*
- ❖ Designing experiments for the AI age
- ❖ Coping with a deluge of mixed-quality contributions

Holding our own with Lean

- ❖ *Using our strengths*
 - ❖ the best automation
 - ❖ our *culture* of legible proofs
 - ❖ scalability to **super gigantic proofs**
 - ❖ no dependent type horrors, e.g. “elaboration”

what we don't do...

The interactions between the components of the elaboration task are subtle, and the challenge is to deal with them all at the same time. A definition or proof may give rise to hundreds of constraints requiring a mixture of higher-order unification, disambiguation of overloaded symbols, insertion of coercions, type class inference, and computational reduction. The net effect is then a difficult constraint-solving problem with a combinatorial explosion of options.

“Elaboration in Dependent Type Theory”, 2015

Embracing simple type theory

- ❖ *Dependent types don't work*, due to intensional equality
- ❖ Carrier sets *when needed* are better than dependent types as global burden (and indexing works!)
- ❖ We still can have p -adic numbers, field extensions, abstract algebra, modules, schemes, ...
- ❖ *Set theory* needed only for theorems about set theory: ordinals, cardinals, order types

Breaking with old ways

Past disruptive steps

- ❖ 1992 moving to *order-sorted polymorphism* (and abandoning the old MLTT-inspired term syntax)
- ❖ 1999 breaking with LCF tradition: *Isar proof language*
- ❖ 2011 goodbye to Proof General: *Prover IDE* and *jEdit*
- ❖ **What's next?**

Mega proofs: ever larger projects

- ❖ Reducing memory usage?
- ❖ Boosting performance and usability?
- ❖ Theory name spaces? Better control over notation?
- ❖ Feature-complete VS Code?
- ❖ Your idea??

Changing of the guard

- ❖ Early developers retired or nearing retirement
- ❖ Withdrawal of facilities at Munich and Cambridge
- ❖ Poly / ML is the sole creation of Dave Matthews
- ❖ **How do we transition to a new generation?**

Acknowledgements

Funders: EPSRC, DFG, ESPRIT, ERC, ...

People: you know the main ones;
the AFP alone has 600 contributors!